element14

GertDuino Board

Exclusively From

GERTBOARD

element14

User Manual

By: G.J. van Loo, Version 1.4

Dated: 6th Nov 2013

1 Contents

Contents

1 **Introduction** 3
1.1 Identify 3
1.2 Comparison 4
1.3 Vext 4
2 **RS232/ UART** 4
2.1 Atmega-328 & Pi UART 5
2.2 Atmega-48 UART 5
3 **Atmega-328** 6
3.1 Features 6
3.2 Program the Atmega-328 6
3.3 Using/running the Atmega-328 7
4 **Atmega-48** 7
4.1 Features 7
4.2 Program the Atmega-48 7
4.3 Using/running the Atmega-48 8
4.4 Real Time Cloc 8
4.5 Infra-red receiver/remote control receiver 8
4.6 Battery Drain 9
4.7 Atmega-48 LED trick 10
5 **Connectors** 11
5.1 Alternate functions. 11
5.2 Atmega-328 13
5.3 Atmega-48 14
5.4 Raspberry-Pi 15
6 **Frequently Asked Questions (FAQs)** 16
7 **How to start** 17
7.1 On the Raspberry-Pi: 17
7.2 On a PC 17
8 **Example programs** 19
8.1 Atmega-328 19
8.2 Atmega-48 23
9 **Control Arduino Reset** 265
10 **Appendix A : GertDuino Schematic** 26

1 Introduction

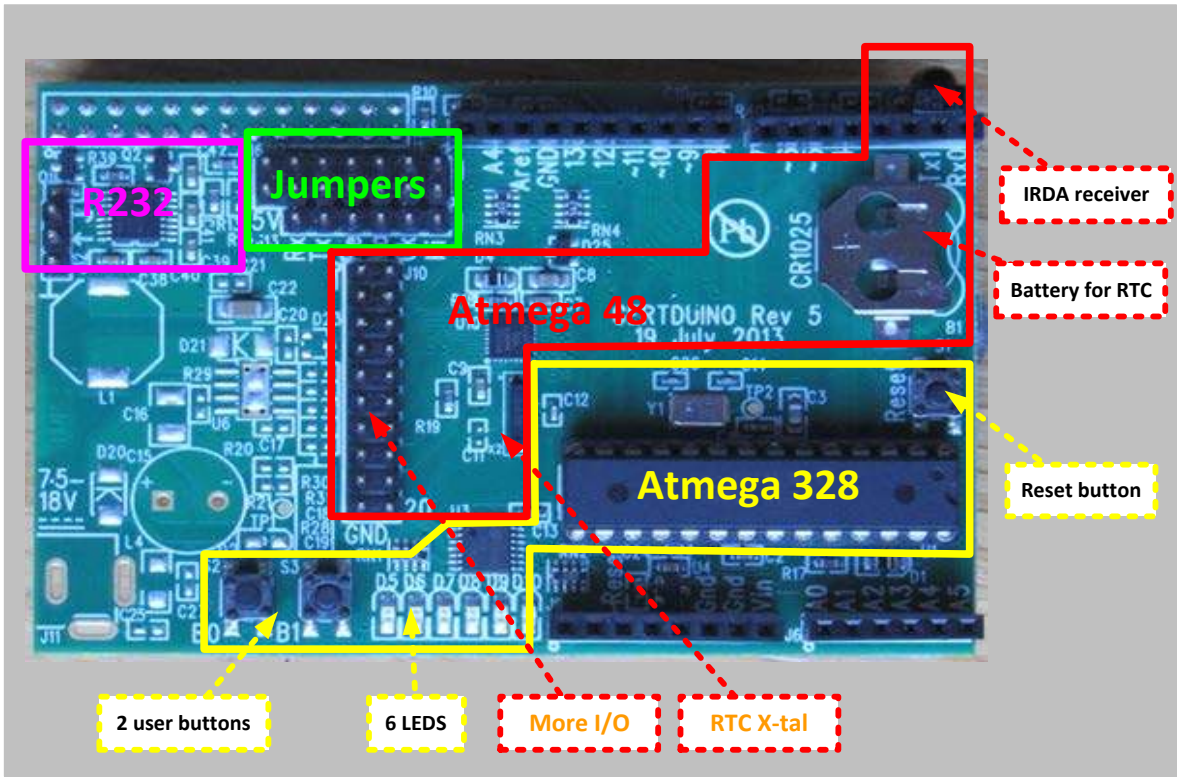
The GertDuino is a Raspberry-Pi add-on board which offers the same functionality as an Arduino-Uno but with some extra added features.

1.1 Identify

The picture below lets you identify the various functions on the board.

- RS232 level converter can be used by:
  - Raspberry-Pi
  - or Atmega-328
  - or Atmega-48
- Atmega 328 (Arduino-Uno® compatible) with:
  - Arduino-Uno® compatible connectors
  - Reset button
  - 2 user push buttons
  - 6 LEDs.
- Atmega 48 with:
  - I/O connector with 20 pins.
  - High precision RTC crystal
  - Battery backup power supply
  - IRDA interface

PCB Overview:



Picture 1: GertDuino Functions

1.2 Comparison

There are some differences between a normal Arduino-Uno and the GertDuino.

Function	Arduino-Uno	GertDuino
USB	Slave interface	-
Reset button	Yes	Yes
Power supply	7..12V, ~250mA	<5V Raspberry-Pi>
3V3 supply	~50mA	~150mA.
LED's	One Not-buffered	Six Buffered
User pushbuttons	-	Two
RS232 buffer	-	Yes
Real-Time-Clock	-	Yes
Infra-red interface	-	Yes

Table 1: Comparison GertDuino vs Arduino-Uno

1.3 Vext

As the board does not have a separate supply the Vext is not connected. If you want it connected you have to add the following components:

J1, L4 (or a short instead of L4), D20 (or a short instead of D20).

2 RS232/ UART

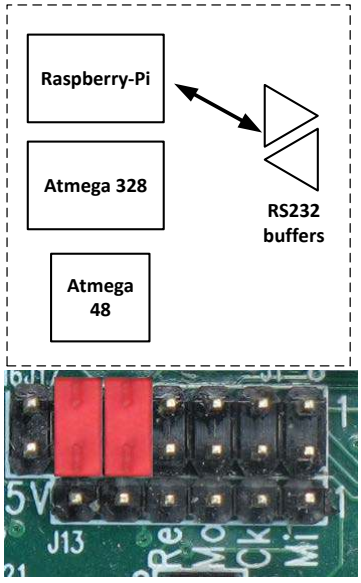
The Gerduino board has a RS232 level converter which will convert the signals form a UART to the RS232 standard voltages (And invert them as per that same standard). The RS232 signals come from J12.

Pin 3 is the receive
Pin 2 is the transmit
Pin 1 is the ground

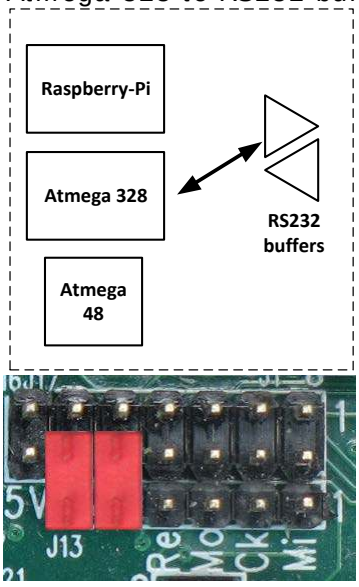
Connections can be made in many ways:

2.1 Atmega-328 & Pi UART

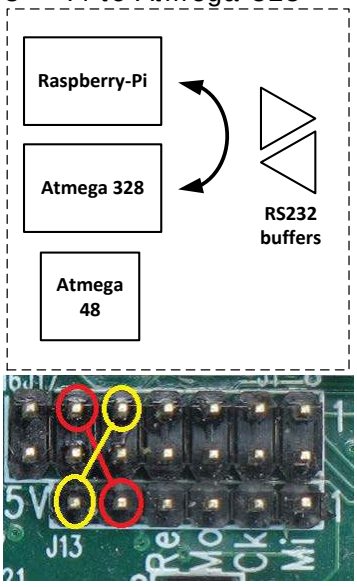
Pi to RS232 buffers



Atmega-328 to RS232 buffers

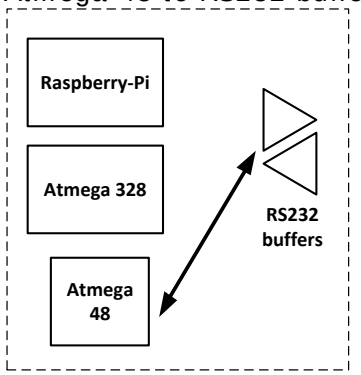


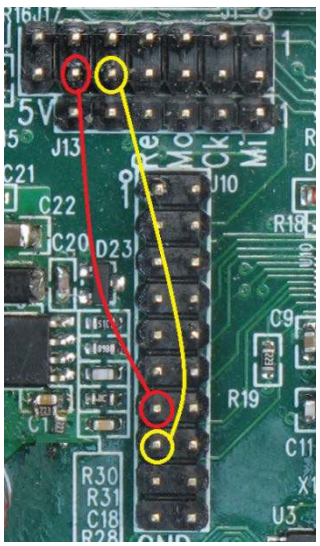
Pi to Atmega-328



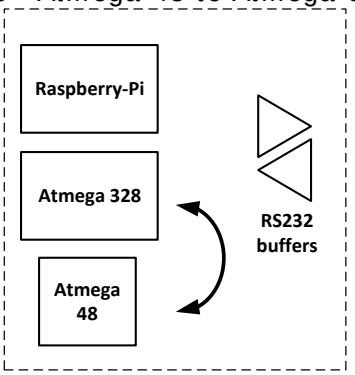
2.2 Atmega-48 UART

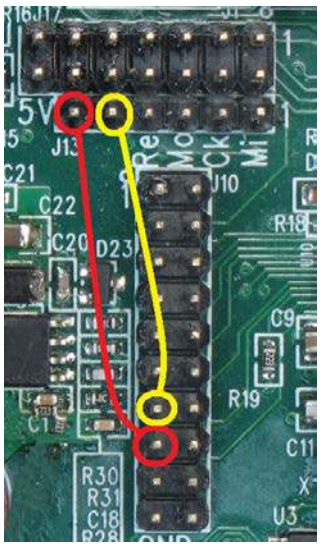
Atmega-48 to RS232 buffers



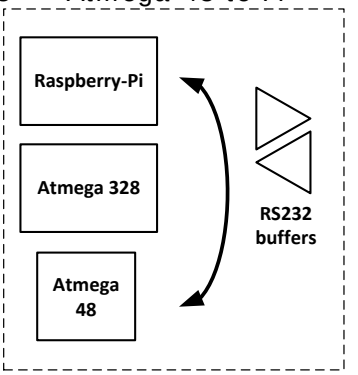


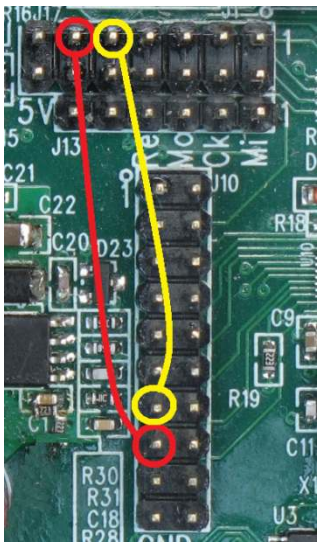
Atmega-48 to Atmega-328





Atmega-48 to Pi





3 Atmega-328

3.1 Features

This device is compatible with the Arduino Uno. In contrast to the 328 on the *GERTBOARD* this device runs of 5V, has the 16MHz oscillator and has connectors which are 100% Arduino-Uno compatible. It also contains the reset switch.

This board also has the following components which you will **not** find on the Uno:

- 2 User push buttons
- 6 LEDs[1]

LEDs

One LED is connected to PB5 (aka Port-13 aka SCK). This is compatible with the UNO. The GertDuino has a five more LEDs. The total list of LEDs is:

- PB5 (Port-13)
- PB1 (Port-9)
- PB2 (Port-10)
- PD3 (Port-3)
- PD5 (Port-5)
- PD6 (Port-6)

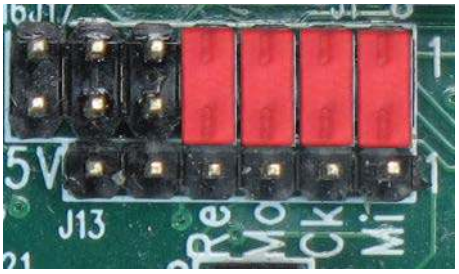
The LEDs are not directly connected but are buffered and thus do not give any significant load on the signal pins.

User buttons

The two user buttons are connected to pins PC2 and PC3. They will only function correctly if the pins have an internal or external pull-up. The button are connected through a 1K Ohm resistor so they will not cause a short if a pin is set as output and the button is pressed.

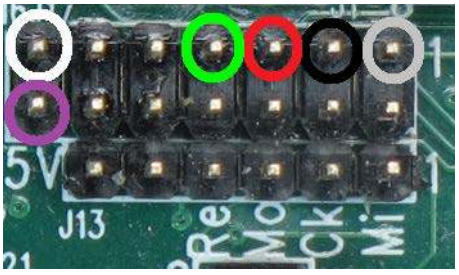
3.2 Program the Atmega-328

To program this device from the Raspberry-Pi you have to place the following 4 jumpers:



Then run the script *Program 328*as described in section *8.1.Atmega-328.*

To program the 328 using a JTAG-ICE you need to use the "squid" cable and make the following connections:



[1] LED: The first debug tool any programmer grabs for.

- At the left there are the GND (white) and 5V (Purple) connections.
- At the top row right are the Reset (green), Mosi (Red), Clk (Black) and Miso (Grey)[2].
- The equivalent JTAG names for these are: nSRST, TDI, TCK, TDO

3.3 Using/ running the Atmega-328

When the device has been programmed it will run that program independent of the Raspberry-Pi. In fact you can remove the board from the Raspberry-Pi and use it standalone.

When developing programs you may leave the jumpers in place as the programme will tri-state its pins and set the reset pin high when it has finished. This is NOT the case if the PI is reset or not powered. Especially the reset-jumper needs to be removed otherwise the Raspberry-Pi GPIO pin 8 (which is default ***low***) will the keep the 328 device in reset or you can run the *reset_off* script.

You should also remove the jumpers if you want to use any of the following pins: B3,B4,B5,C6.

4 Atmega-48

4.1 Features

This device is intended to be used as Real Time Clock (RTC) and/or as IRDA front end. However it is also freely programmable by the user and thus can be used for any other application, giving the user the power of not one but TWO Atmega devices to play with.

Note: *The I2C interface of the Atmega-48 is connected permanently to the Raspberry-Pi I2C interface <GOIO0/1 on rev1, GPIO 2/3 on rev2>.*

Also beware that if you make programming errors with the Atmege-328 the device can easily be replaced. This is **not** the case with the Atmega-48. It is therefore strongly recommended that you are extra careful and do not damage any of the I/O ports.

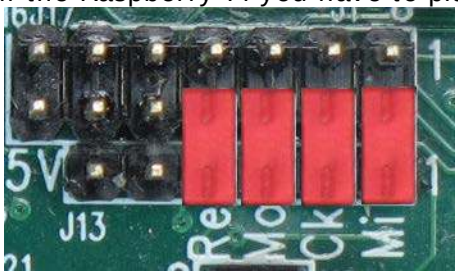
Spare connections.

The following I/O pins of the Atmega-48 are not used and are brought out to a connector: B0,B1,B2, B3, B4,B5, C0,C1,C2,C3,D0,D1,D4, D5, D6, D7.

Beware that B3, B4 and B5 are also used for programming the device.

4.2 Program the Atmega-48

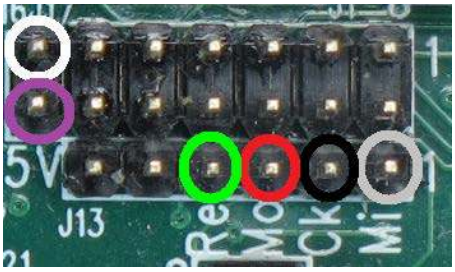
To program the Atmega-48 from the Raspberry-Pi you have to place the following 4 jumpers:



The programming is the same as the 328 but replace "328p" with "48pa".

[2] The colours used here are the same as on MY squid cable but I can't guarantee all squid cables are the same.

To program the 48 using a JTAG-ICE you need to use the "squid" cable and make the following connections:



- At the left there are the GND (white) and 5V (Purple) connections.
- At the bottom row right are the Reset (green), Mosi (Red), Clk (Black) and Miso (Grey).[3]
- The equivalent JTAG names for these are: nSRST, TDI, TCK, TDO

4.3 Using/ running the Atmega-48

What was written about the 328 also is valid for the 48: when the device has been programmed it will run that program independent of the Raspberry-Pi. In fact you can remove the board from the Raspberry-Pi and use it standalone.

When developing programs you may leave the jumpers in place as the programme will tri-state its pins and set the reset pin high when it has finished. This is NOT the case if the PI is reset or not powered. Especially the reset-jumper needs to be removed otherwise the Raspberry-Pi GPIO pin 8 (which is default ***low***) will the keep the 48 device in reset or you can run the *reset_off* script.

You should also remove the jumpers if you want to use any of the following pins: B3,B4,B5,C6.

4.4 Real Time Clock

The Atmege-48 device has a 32768Hz crystal connected to operate as a Real-Time-Clock (RTC). Example code for this can be found under section ***8.2 Atmega-48***. The Crystal is a high quality type and under normal conditions a deviation is less than 1 sec/3 days.

The other part of the RTC is that the Atmega-48 has a 3V battery. It will switch to that battery when the 5V power is removed. As the Atmega-48 is a fully programmed microcontroller it can be set-up to perform other operations or hold other data when the main power of the BCM2835 is removed.

If you have programmed the Atmege-48 correctly it uses ~1µA when powered down.

4.5 Infra-red receiver/ remote control receiver

The BCM2835 does not have a native IRDA interface. The protocol can be implemented using a standard GPIO pin but that puts a very heavy burden on the CPU. To support IRDA the Atmega-48 has a TSSOP4038 IRD device connected to pin D3. This device supports the most common IRDA protocol: 38KHz IR signal.

Unfortunately we could not run the IRDA interface from the battery as it uses too much current (~450 µA). Thus you need the 5V present for it to operate.

[3] The colours used here are the same as on MY squid cable but I can't guarantee all squid cables are the same.

The IRDA can also be used if the GertDuino is used stand-alone to control the connected logic using a remote control.

Note that 95% of all TV/Video/CD remote controls use the 38KHz infra-red signal, but the coding varies greatly from type to type.

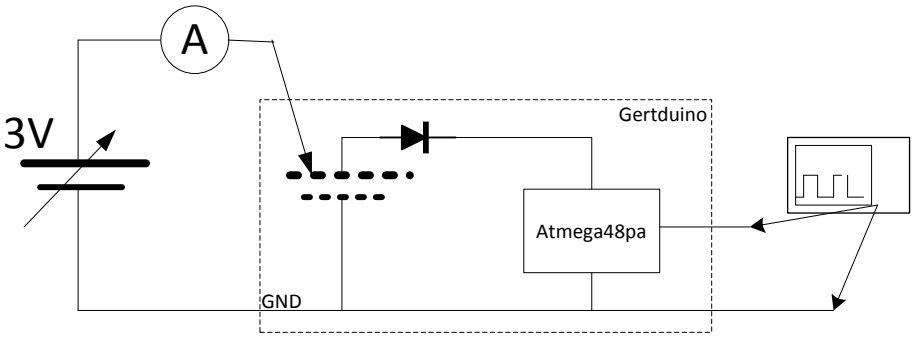
4.6 Battery Drain

If a battery is present and the power of the Raspberry-Pi is switched of the Atmega-48 will still remain powered by the Battery. It will also keep running. Unless the battery is removed or the Atmega-48 is programmed to go into a special ultra-low-power condition, the battery will be drained in a short time.

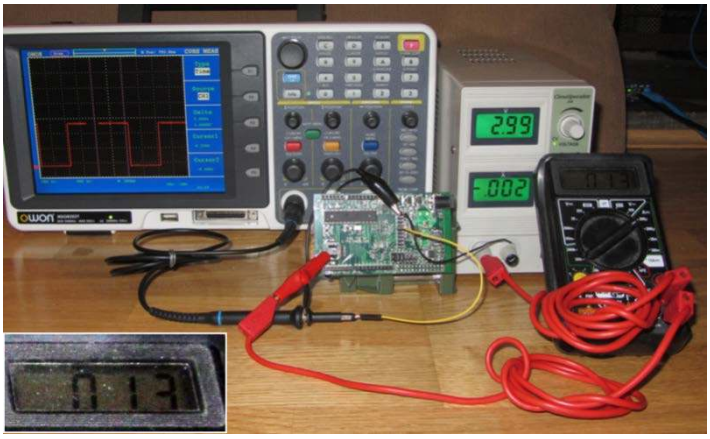
Even if you think the device is in ultra-low-power mode it can still consume power if it has to drive outputs high.

Measurements have also show that if a UART connection exists between the Atmega-48 to the Raspberry-Pi (even if it is not used) that increases the lower power current from 1µA to about 100µA.

To measure the current consumption you have to use a 3V supply and connect it to the battery holder but between the supply and the battery holder you have to place a current (Ampere) meter. Optionally you can connect a scope on one of the I/O pins of the Atmege48 to see if the program is running. You should NOT connect anything to one of the output which loads an I/O pin as that will cause extra current consumption.
This is a block diagram of the setup:



And this is how it looks in real life:



The meter shows a current consumption of 1.3µAmp. (The meter is shown enlarged in the lower left hand corner of the picture) .

If possible limit the current from your power source to a few milli-amps. I managed to blow a fuse of my meter performing the measurements because I accidentally shorted the supply when placing the probe on the battery holder.

4.7 Atmega-48 LED trick

If you are debugging, an LEDs is often the first tool you reach for. But the Atmega-48 does not have any LEDs. However the Atmega-328 does! There are two ways in which you can use these LEDs :

- The safest way is to remove the 328 from its socket.
- The second way is to erase the 328 so that all its pins are inputs.

You can then use the connectors to feed a signal to an LED. Simplest way is to use a female-male strap between connector J10 and e.g. pins, 2,3 or 6 of J14.

5 Connectors

The board contains a number of connectors. You will find that in the document the connectors of the Atmega devices have two ways of numbering: There are the single numbers 0..13 and A1..A3. These are the numbers used in many Arduino example programs. Alongside those I use the official pin names (PB0..PB7, PD0..PD7, PCO..PC3). The latter are easier to use if you have to work with the AVR datasheet.

5.1 Alternate functions.

The Atmega-328 and the Atemege-48 have exactly the same pins with the same functionality. The devices only differ in the size of their various memories. The following is a table of the pins and all the functions they can carry. These where copied from the AVR datasheet. For details of the functions you should read that datasheet.

#	Name	Functions
-	PB7	XTAL2 (Chip Clock Oscillator pin 2) TOSC2 (Timer Oscillator pin 2) PCINT7 (Pin Change Interrupt 7)
-	PB6	XTAL1 (Chip Clock Oscillator pin 1 or External clock input) TOSC1 (Timer Oscillator pin 1) PCINT6 (Pin Change Interrupt 6)
13	PB5	SCK (SPI Bus Master clock Input) PCINT5 (Pin Change Interrupt 5)
12	PB4	MISO (SPI Bus Master Input/Slave Output) PCINT4 (Pin Change Interrupt 4)
11	PB3	MOSI (SPI Bus Master Output/Slave Input) OC2A (Timer/Counter2 Output Compare Match A Output) PCINT3 (Pin Change Interrupt 3)
10	PB2	SS (SPI Bus Master Slave select) OC1B (Timer/Counter1 Output Compare Match B Output) PCINT2 (Pin Change Interrupt 2)
9	PB1	OC1A (Timer/Counter1 Output Compare Match A Output) PCINT1 (Pin Change Interrupt 1)
8	PB0	ICP1 (Timer/Counter1 Input Capture Input) CLKO (Divided System Clock Output) PCINT0 (Pin Change Interrupt 0)

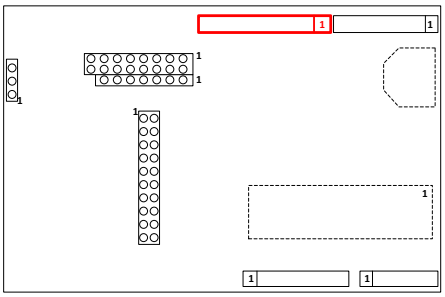
#	Name	Functions
A5	PC5	ADC5 (ADC Input Channel 5) SCL (2-wire Serial Bus Clock Line) PCINT13 (Pin Change Interrupt 13)
A4	PC4	ADC4 (ADC Input Channel 4) SDA (2-wire Serial Bus Data Input/Output Line) PCINT12 (Pin Change Interrupt 12)
A3	PC3	ADC3 (ADC Input Channel 3) PCINT11 (Pin Change Interrupt 11)
A2	PC2	ADC2 (ADC Input Channel 2) PCINT10 (Pin Change Interrupt 10)
A1	PC1	ADC1 (ADC Input Channel 1) PCINT9 (Pin Change Interrupt 9)
A0	PC0	ADC0 (ADC Input Channel 0) PCINT8 (Pin Change Interrupt 8)

#	Name	Functions
7	PD7	AIN1 (Analog Comparator Negative Input) PCINT23 (Pin Change Interrupt 23)
6	PD6	AIN0 (Analog Comparator Positive Input) OC0A (Timer/Counter0 Output Compare Match A Output) PCINT22 (Pin Change Interrupt 22)
5	PD5	T1 (Timer/Counter 1 External Counter Input) OC0B (Timer/Counter0 Output Compare Match B Output) PCINT21 (Pin Change Interrupt 21)
4	PD4	XCK (USART External Clock Input/Output) T0 (Timer/Counter 0 External Counter Input) PCINT20 (Pin Change Interrupt 20)
3	PD3	INT1 (External Interrupt 1 Input) OC2B (Timer/Counter2 Output Compare Match B Output) PCINT19 (Pin Change Interrupt 19)
2	PD2	INT0 (External Interrupt 0 Input) PCINT18 (Pin Change Interrupt 18)
1	PD1	TXD (USART Output Pin) PCINT17 (Pin Change Interrupt 17)
0	PD0	RXD (USART Input Pin) PCINT16 (Pin Change Interrupt 16)

5.2 Atmega-328

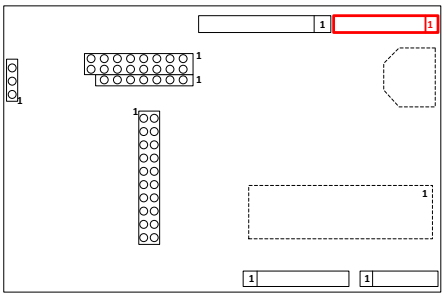
The Atmega-328 pins are brought to connectors compatible with the Arduino-Uno.

J14



Pin No.	Signal
10	A5/PC5/SCL
9	A4/PC4/SDA
8	AREF
7	Ground
6	13/PB5/SCK/**LED0**
5	12/PB4/MISO
4	11/PB3/MOSI
3	10/PB2/SS/**LED2**
2	9/PB1/PCINT1/**LED1**
1	8/PB0/CLK0

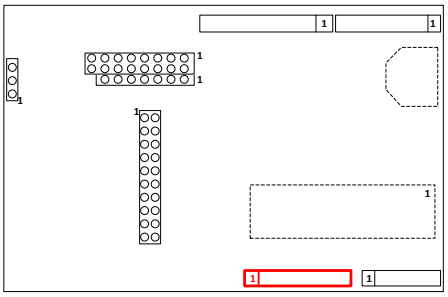
J7



Pin No.	Signal
8	7/PD7/AIN1
7	6/PD6/AIN0/**LED6**
6	5/PD5/T1/**LED5**
5	4/PD4/T0
4	3/PD3/INT1/**LED4**
3	2/PD2/INT0
2	1/PD1/TXD
1	0/PD0/RXD

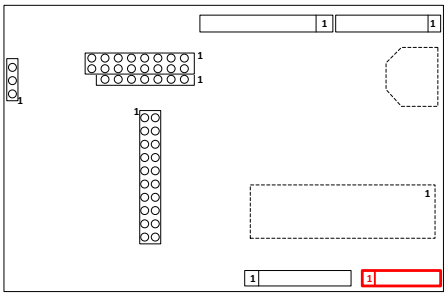
Pin 1 is on the right-hand side so these tables top-to-bottom are the pins from ***left-to-right***.

J9



Pin No.	Signal
8	**NC**
7	Ground
6	Ground
5	5V
4	3V3
3	Reset (Active low)
2	5V
1	NC

J6



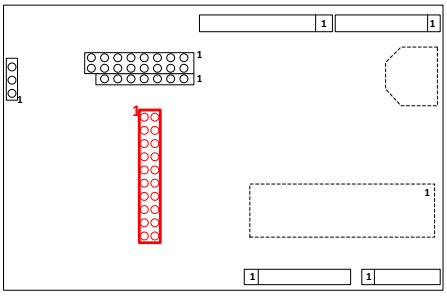
Pin No.	Signal
6	A5/PC5/SCL
5	A4/PC4/SDA
4	A3/PC3/ADC3/**BUT1**
3	A2/PC2/ADC2/**BUT0**
2	A1/PC1/ADC1
1	A0/PC0/ADC0

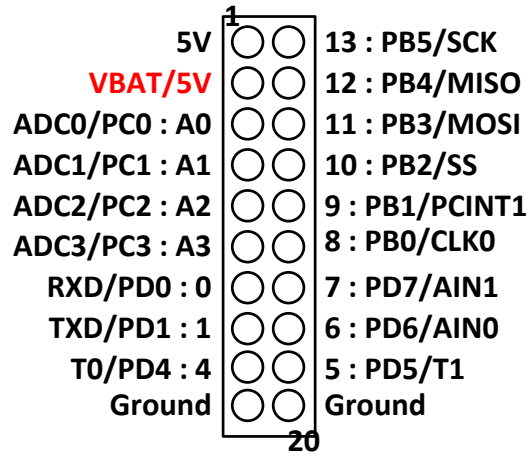
Pin 1 is on the left-hand side so these tables top-to-bottom are the pins from ***right-to-left***.

Beware that Pin 8 of J9 is normally connected directly to the input voltage which has been removed and thus is NC here.

5.3 Atmega-48

All unused pin of the Atmega-48 are brought out to a 20 -pin connector:





The supply 5V/VBAT which goes to the Atmega-48 also goes to the connector pin 3. Any equipment connected to that pin will also draw current from the battery if the 5V is switched off.

The supply comes through a Schottky diode so the actual voltage is lower: ~4.5 Volts. Also the current consumption should be limited ~100mA.

The following pins of the ATmega-48 are dedicated connected:

Pin	Hard wired to	Function
PD2	5V Supply	Detect absence of 5V supply (for RTC)
PD3	IRDA output	Receive IRDA signal
PC5	SCL	I^2C connection with the Pi
PC4	SDA	I^2C connection with the Pi
PB7	XTAL1	32768Hz Tuning crystal
PB6	XTAL2	32768Hz Tuning crystal
PC6	Program reset	Reset when programming

The Atmege-48 does not have a dedicated reset pin as that would interfere with its function as real-time-clock. A reset can be obtained by pulling pin 4 of J13 low.

5.4 Raspberry-Pi

All connections between the board and the Raspberry-Pi are protected against 5V signals. The I^2C bus has FET level switches. All the other signals use resistive dividers.
The following connections of the Raspberry-Pi are used:

- 5V
- 3V3 (I^2C level converters only)
- GPIO0/2 (I^2C SDA)
- GPIO1/3 (I^2C SCL)

The following connections of the Raspberry-Pi are used if the programming jumpers or UART jumpers are placed:

- GPIO14 (UART-Tx)
- GPIO15 (UART-Rx)
- GPIO8 (Reset)
- GPIO9 (MISO)
- GPIO10 (MOSI)
- GPIO11 (SCLK)

6 Frequently Asked Questions (FAQs)

Some questions you may ask and the answers.

avrdude: AVR device not responding

Q: When I try to program the device I get an error: "avrdude: AVR device not responding."

A: The most likely cause is that you have forgotten to place the four programming jumpers. See section *3.2 Program the Atmega-328.*

Why is my program slow?

Q: When I run the program it is very slow. Where I expect a delay of 1second it takes much longer.

A: Straight from the factory the CPU runs from the internal 8MHz clock and that is divided by 8. Thus the processor runs at 1 MHz. To switch to the full speed, using the external 16MHz oscillator run the avrdude command as described in *8.1Atmega-328*under "**Initial clock setup"**

Why does my program not run?

Q: When I upload the program it runs fine but when I halt the Raspberry-Pi or when I start the Raspberry-Pi my program does not work.

A: GPIO 8 controls the Reset of the Arduino. This pins must be high but for your program to run. The simplest solution is to remove the programming jumpers. Alternative is to program the GPIO-8 pin high using the *reset_off* script. The avrdude with the -c gpio option does this for you so normally after programming the reset has been removed.

I have a different compiler

Q: I use the AVR compiler on my PC. How do I program the Atmega on the Raspberry-Pi?

A: I have only experience with the GCC version (AVR 5.1 and higher). After compilation you find a **.hex** file in the **debug** directory. You have to transfer that file somehow to the Raspberry-Pi and use the programmer script **Program 328** as described in *8.1Atmega-328* to program the device(s) on the GertDuino. (If you have the script already installed use "./program_328 <hex file>)

The Raspberry-Pi boots different: it has big text and not the normal prompt!

Q: When I plug the GertDuino on the Raspberry-Pi it boots different: It has big text and not the normal prompt!

A: Pin 5 of the GPIO connector is used to indicate 'safe boot mode'. If that pin is low when booting the Raspberry-Pi boots in "safe mode". Pin 5 is also connected to the Atmge-48. It is one of the I2C pins. Thus if your 48 is driving a LOW on that pin the Pi always boots in safe mode.
To prevent this you can put "avoid_safe_mode=1" in the config.txt file and the pi will boot normally.

Why is there no battery supplied

Q: The GertDuino has a battery holder but there is no battery in there. Why do I have to buy my own?

A: These batteries are lithium batteries. Those are classified as '***Dangerous Goods'*** and require special paper work, warning labels and other precautions when shipped. And that is for shipping within the UK. International shipping becomes a nightmare. So we decided to leave it off.

7 How to start

Before you can program the devices you need to have a cross compiler. A cross compiler is a compiler which runs on one type of processor, but generates code for a different type. In this case the compiler runs on the Raspberry-Pi (ARM11 device) but makes code for the Atmel devices.

7.1 On the Raspberry-Pi:

When programming the Atmel devices on the Raspberry-Pi you have two choices:

- Use the Arduino GUI
- Use the GCC Atmel compiler

For both you need to have a cross compiler for the Atmega devices. Easiest is to install the Arduino package:

```
sudo apt-get install arduino
```

avrdude

You need to use a program called "*avrdude*" to program the devices BUT you need a special version of "*avrdude*" which can program the devices using the GPIO of the Raspberry-Pi. Thanks for Gordon Henderson (projects.drogon.net) who has provided these:

Standard Debian Squeeze:

```
cd /tmp
wget http://project-downloads.drogon.net/gertboard/avrdude_5.10-4_armel.deb
sudo dpkg -i avrdude_5.10-4_armel.deb
sudo chmod 4755 /usr/bin/avrdude
```

Debian Raspbian:

```
cd /tmp
wget http://project-downloads.drogon.net/gertboard/avrdude_5.10-4_armhf.deb
sudo dpkg -i avrdude_5.10-4_armhf.deb
sudo chmod 4755 /usr/bin/avrdude
```

You can now compile programs for the Atmega devices and upload the program into the chip on the GertDuino. Example source code, Makefile and how to upload the program can all be found in section ***8 Example programs***.

If you want to use the Arduino development environment you have to adapt it. See projects.drogon.net/raspberry-pi/gertboard/arduino-ide-installation-isp/ how to do that.

7.2 On a PC

Atmel have a free C-compiler. You can get information about the latest version here:
http://www.atmel.com/tools/ATMELSTUDIO.aspx

You can compile on the PC but you need to transfer the final *.hex* file to the Raspberry-Pi before you can program the Atmega devices.

Alternative is that you buy a JTAG-ICE box and use that to program and the devices but that is a lot more expensive. It does have the advantage that you can use it for debugging as well: Step through the program, set breakpoints ,inspect variables etc.

8 Example programs

8.1 Atmega-328

blink.c source code:

```
/*
 * blink.c
 *
 * Created: 23/09/2013 21:04:02
 * Author: G.J. van Loo
 * Simple example program to 'walk' the LEDs
 */

#include <avr/io.h>

#define DELAY 250
#define F_CPU 16000000

// Some macros that make the code more readable
#define output_low(port,pin) port &= ~(1<<pin)
#define output_high(port,pin) port |= (1<<pin)
#define set_input(portdir,pin) portdir &= ~(1<<pin)
#define set_output(portdir,pin) portdir |= (1<<pin)

// Outputs are:
// LED0 = PB5
// LED1 = PB1
// LED2 = PB2
// LED3 = PD3
// LED4 = PD5
// LED5 = PD6


void delay_ms(unsigned int ms)
{
  uint16_t delay_count = F_CPU / 17500;
  volatile uint16_t i;

  while (ms != 0) {
    for (i=0; i != delay_count; i++);
      ms--;
  }
} // delay_ms

void delay()
{ long d;
   unsigned char oldb,oldd;
   for (d=0; d<DELAY; d++)
   {
     delay_ms(1);
     if ((PINC & 0b00001000)==0)
     {
       oldb = PORTB;
```

```
       oldd = PORTD;
       PORTB = 0xFF;
       PORTD = 0xFF;
       delay_ms(1);
       PORTB = oldb;
       PORTD = oldd;
       d--;
     }
     else
     { if ((PINC & 0b00000100)==0)
          d--;
       else
         delay_ms(1);
     }   // if button pressed
   }  // if button pressed
} // delay

int main(void)
{  // int b;
    // Set all LED connections to output
    DDRB = 0b00100110;
    DDRD = 0b01101000;
    PORTB = 0x00;
    PORTD = 0x00;
    // Set button (port C) to input
    DDRC = 0b00000000;
    // pull-up on C2 & C3:
    PORTC = 0b00001100;

    while(1)
    { // convoluted but simple walk the leds
        output_high(PORTB,5);
        delay();
        output_low (PORTB,5);
        output_high(PORTB,1);
        delay();
        output_low (PORTB,1);
        output_high(PORTB,2);
        delay();
        output_low (PORTB,2);
        output_high(PORTD,3);
        delay();
        output_low (PORTD,3);
        output_high(PORTD,5);
        delay();
        output_low (PORTD,5);
        output_high(PORTD,6);
        delay();
        output_low (PORTD,6);
        output_high(PORTD,5);
        delay();
        output_low (PORTD,5);
        output_high(PORTD,3);
        delay();
        output_low (PORTD,3);
        output_high(PORTB,2);
```

```
        delay();
        output_low (PORTB,2);
        output_high(PORTB,1);
        delay();
        output_low (PORTB,1);
    } // forever
} // main
```

Makefile

```
# Makefile:
#     Make the GertDuino m328p firmware.
#
# Copyright (c) 2013 Gordon Henderson <projects@drogon.net>
#################################################################################
# This file is part of gertduino-m328:
#Software to run on the Atmega328p processor on the Gerduino board
#Can be used for the Atmega328p processor on the GERTBOARD as well
#    This is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this.  If not, see <http://www.gnu.org/licenses/>.
#################################################################################

TARGET=blink

MCU=atmega328p
FREWQ=16000000

# Debug
#DEBUG      = -gstabs

# C flags
CC    = avr-gcc
#CFLAGS = $(DEBUG) -O3 -Wall -std=gnu99 -mmcu=$(MCU) -DF_CPU=$(FREWQ) $(INCLUDE)
CFLAGS = $(DEBUG) -O2 -mcall-prologues -Wall -std=gnu99 -mmcu=$(MCU) -
DF_CPU=$(FREWQ) $(INCLUDE)

LD    = avr-gcc
#LDFLAGS2=-Wl,-uvfprintf -lprintf_flt
LDFLAGS = -mmcu=$(MCU) $(DEBUG) $(LIBLOC) $(LDFLAGS2)
#LIBS    = -ldross -lm
SRC   =     $(TARGET).c
OBJ   =     $(SRC:.c=.o)

all:  $(TARGET).hex

$(TARGET).hex: $(TARGET).elf
      @echo [hex] $<
```

```
	@avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex

$(TARGET).elf:    $(OBJ)
	@echo [Link] $<
	@$(LD) -o $@ $(OBJ) $(LDFLAGS) $(LIBS)
	@avr-size $(TARGET).elf

# Generate .lst file rule
%.lst : %.o
	@echo [lst] $<
	@avr-objdump -h -S $<> $@

.c.o:
	@echo [CC] $<
	@$(CC) -c $(CFLAGS) $< -o $@

.PHONEY:    clean
clean:
	rm -f *.o *.elf *.hex *.lst Makefile.bak *~
```

Program 328

```
#!/bin/bash
# script to program 328p device using AVRDUDE and a hex file
if [ "$1" == "" ]; then echo Missing argument
  exit 1;
fi
# if ends in .hex use full argument
# otherwise add the .hex
ext=${1:${#1}-4}
if [ "$ext" == ".hex" ]; then
  /usr/bin/avrdude -c gpio -p m328p $1 -Uflash:w:$1
else
  /usr/bin/avrdude -c gpio -p m328p $1.hex -Uflash:w:$1.hex
fi
```

Save the above code in a file called program_328 and then run "*chmod 777 program_328*". Use *./program_328 <hex file>* to program the Atmega device.

Initial clock setup

```
avrdude -qq -c gpio -p atmega328p -U lock:w:0x3F:m    -U efuse:w:0x07:m -
U lfuse:w:0xE7:m -U hfuse:w:0xD9:m
```

You normally run the above command when you get a brand new device. It programs the Atmega328 to use the external 16MHz Crystal.

8.2 Atmega-48

This section shows an example program for the Atmega48. You will find that the makefile and the programming files are very similar to the 328 example.

low_power.c source code:

```
// Example code which uses the 32767KHz
// Crystal to implement a 1-second event
// handler
//
// Atmega Low power operation example
// Using a 32768 Khz crystal on timer 2 and full power down mode
// to implement a 1-second event handler
//
// This code is written for the GCC compiler
// Example for the GertDuino Atmega 48PA device
// (This program will NOT run on the 328!)
// This code is freeware
//

#include <avr/interrupt.h>
#include <avr/sleep.h>

volatile unsigned long count_seconds;

main()
{
  // set PB0 as output
  DDRB = 0xFE;

// Set-up 32 KHz oscillator
  TIMSK2 = 0x00;     // No interrupts
  ASSR   = 0x20;     // async run from xtal
  TCNT2  = 0;        // clear counter
  TCCR2B = 0x05;     // prescale 5=/128

  // Wait for all 'busy' bits to be clear
  // That happens on the first timer overflow
  // which can take 8 seconds if you have a max pre-scaler!!
  while (ASSR&0x07) ;

  TIMSK2 = 0x01;     // overflow IRQ enable

  count_seconds = 0; // clear seconds counter
  sei();             //set the Global Interrupt Enable Bit

  while (1)
  {
     SMCR = 0x7; // Go into lowest power sleep mode
     asm("sleep");
     asm("nop");
     // Interrupt woke us up
     // If we get here the interrupt routine has already been called

     // Toggle LED on port B0 using LS timer bit
     PORTB = count_seconds & 0x01;
  }
} // main

//
```

```
// Timer 2 overflow
// if we set timer2 up correctly this routine is called every second
//
ISR(TIMER2_OVF_vect)
{ count_seconds++; // all we do here is count seconds elapsed
}
```

Makefile:

```
# Makefile:
# Make the GertDuino m48p firmware.
#
# Copyright (c) 2013 Gordon Henderson <projects@drogon.net>
#################################################################################
# This file is part of gertduino-m328:
#Software to run on the Atmega328p processor on the Gerduino board
#Can be used for the Atmega328p processor on the GERTBOARD as well
#    This is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this.  If not, see <http://www.gnu.org/licenses/>.
#################################################################################

TARGET=low_power

MCU=atmega48p
FREWQ=1000000

# Debug
#DEBUG      = -gstabs

# C flags
CC    = avr-gcc
#CFLAGS     = $(DEBUG) -O3                  -Wall -std=gnu99 -mmcu=$(MCU) -
DF_CPU=$(FREWQ) $(INCLUDE)
CFLAGS      = $(DEBUG) -O2 -mcall-prologues -Wall -std=gnu99 -mmcu=$(MCU) -
DF_CPU=$(FREWQ) $(INCLUDE)

LD    = avr-gcc
#LDFLAGS2=-Wl,-uvfprintf -lprintf_flt
LDFLAGS = -mmcu=$(MCU) $(DEBUG) $(LIBLOC) $(LDFLAGS2)
#LIBS    = -ldross -lm
SRC   =     $(TARGET).c
OBJ   =     $(SRC:.c=.o)

all:  $(TARGET).hex

$(TARGET).hex: $(TARGET).elf
      @echo [hex] $<
      @avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex
```

```
$(TARGET).elf:    $(OBJ)
      @echo [Link] $<
      @$(LD) -o $@ $(OBJ) $(LDFLAGS) $(LIBS)
      @avr-size $(TARGET).elf

# Generate .lst file rule
%.lst : %.o
      @echo [lst] $<
      @avr-objdump -h -S $<> $@

.c.o:
      @echo [CC] $<
      @$(CC) -c $(CFLAGS) $< -o $@

.PHONEY:    clean
clean:
      rm -f *.o *.elf *.hex *.lst Makefile.bak *~
```

Program 48

```
#!/bin/bash
# script to program 48pa device using AVRDUDE and a hex file
if [ "$1" == "" ]; then
  echo Missing argument
  exit 1;
fi
# if ends in .hex use full argument
# otherwise add the .hex
ext=${1:${#1}-4}
if [ "$ext" == ".hex" ]; then
  /usr/bin/avrdude -c gpio -p m48p $1 -Uflash:w:$1
else
  /usr/bin/avrdude -c gpio -p m48p $1.hex -Uflash:w:$1.hex
fi
```

Save the above code in a file called "*program_48*" and then run "*chmod 777 program_48*". Use *./program_48 <hex file>* to program the Atmega 48 device.

9 Control Arduino Reset

The Raspberry-Pi GPIO 8 pin controls the Arduino reset pin when the jumpers are in place. When starting the pin is LOW and thus the Arduino chip is held in reset. To control the reset (gpio-8 pin) you can use the scripts shown below.

Don't forget to change the mode of the text file to executable format: (chmod 777 reset_off). Depending on your path you may have to call the script starting with a <dot><slash>: "*./reset_off*".

Alternative copy the scripts to */usr/bin: "sudo cp reset_off /usr/bin"*. If you want the Raspberry Pi to always execute the script at boot up you have to edit the */etc/rc.local* file. Make sure that you have the full path in there. Thus if you have installed the script in */usr/bin* you have to add the following line to /etc/rc.local:

/usr/bin/reset_off

reset_off

The following script will release the Arduino reset and thus make that the Arduino chip runs. It only works if the GertDuino is plugged in to the Raspberry Pi and the reset jumper is in place.

```
#!/usr/bin/sudo bash
# Set GPIO pin 8, high  releasing Arduino reset
sudo echo "8"   >/sys/class/gpio/export
sudo echo "out" >/sys/class/gpio/gpio8/direction
sudo echo "1"   >/sys/class/gpio/gpio8/value
sudo echo "8"   >/sys/class/gpio/unexport
```

reset_on

The following script will assert the Arduino reset and thus make that the Arduino chip stops, is held in reset. It only works if the GertDuino is plugged in to the Raspberry Pi and the reset jumper is in place.

```
#!/usr/bin/sudo bash
# Set GPIO pin 8, low activating Arduino reset
sudo echo "8"   >/sys/class/gpio/export
sudo echo "out" >/sys/class/gpio/gpio8/direction
sudo echo "0"   >/sys/class/gpio/gpio8/value
sudo echo "8"   >/sys/class/gpio/unexport
```

10 Appendix A : GertDuino Schematic